Sissejuhatus infotehnoloogiasse

6. loeng: programmid protsessoril

Ülevaade loengust

- Tuletame korraks meelde Turingi masinat
- Reaalne riistvara:
  - Kiire meeldetuletus: riistvara põhikomponendid
  - Paar sõna protsessori tööpõhimõtete osas
  - Mälu hierarhia
- Programmeerimiskeelte hierarhia ja mehaanika (jätkub järgmine loeng)
  - Masinkood
  - Assembler
  - C

Vaatame uuesti Turingi masina simulaatorit

Turingi masina mälu on lint, mida loeb/kirjutab eraldi pea.
Programm on eraldi tabelis ridadena, mis kõik on kujul:

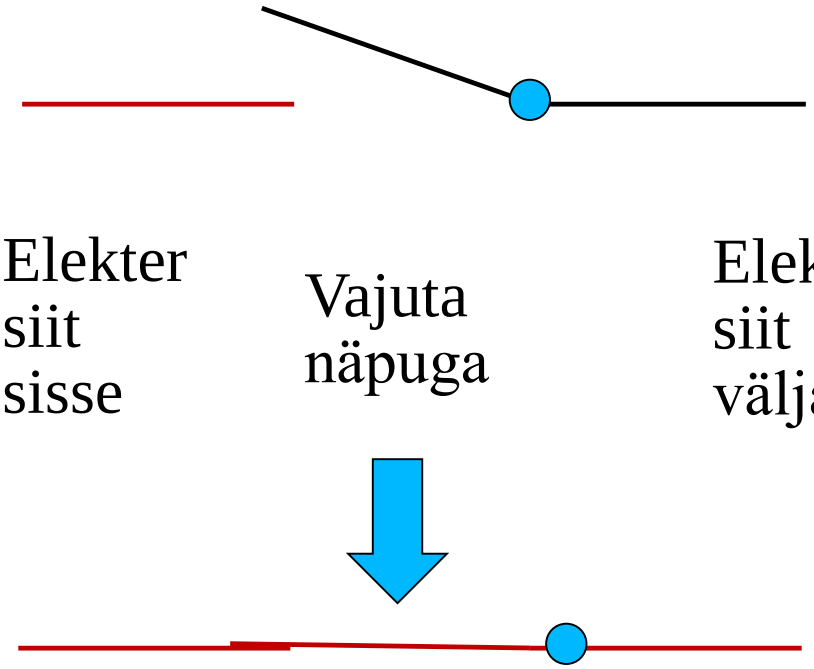
<current state> <current symbol> <new symbol> <direction> <new state>

http://morphett.info/turing/turing.html

Reaalsetel arvutitel saab lugeda/kirjutada mälu suvalisest 8-ga jaguvast kohast minimaalselt 8 bitti (üks bait) korraga. Mõnel protsessoril saab lugeda ainult 32-ga jaguvast kohast 4 baiti korraga.

Käske on protsessoril palju ning nad asuvad sellessamas mälus.

Näpuga lüliti



Vool läbi ei lähe

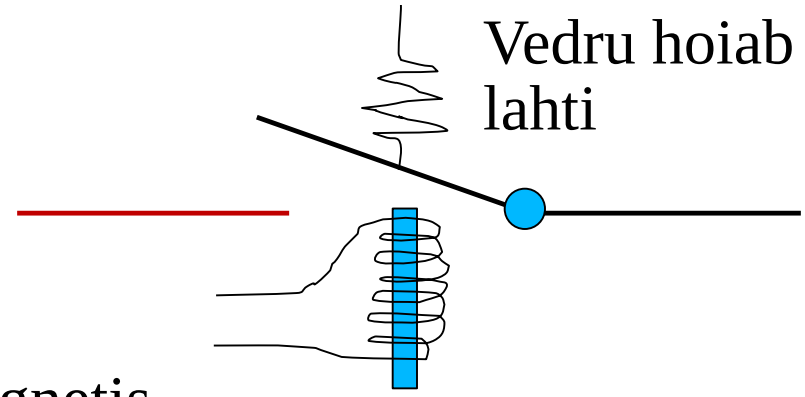
Elekter siit sisse

Vajuta näpuga

Elekter siit välja?

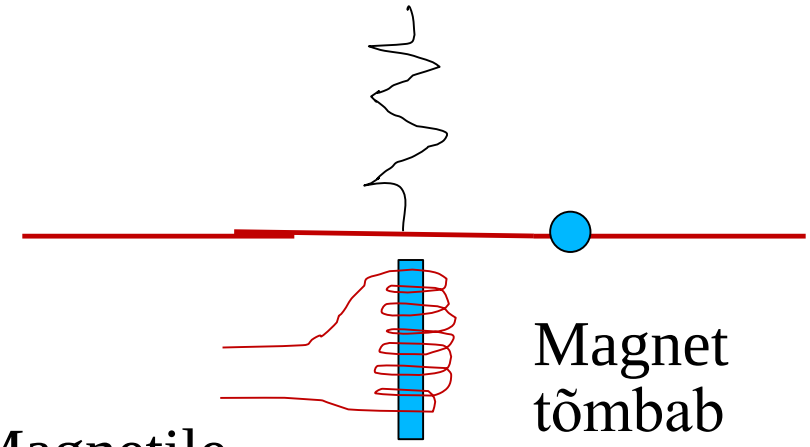
Vool läheb läbi

Relee: mootoriga lüliti



Vool läbi ei lähe

Magnetis
ei ole voolu

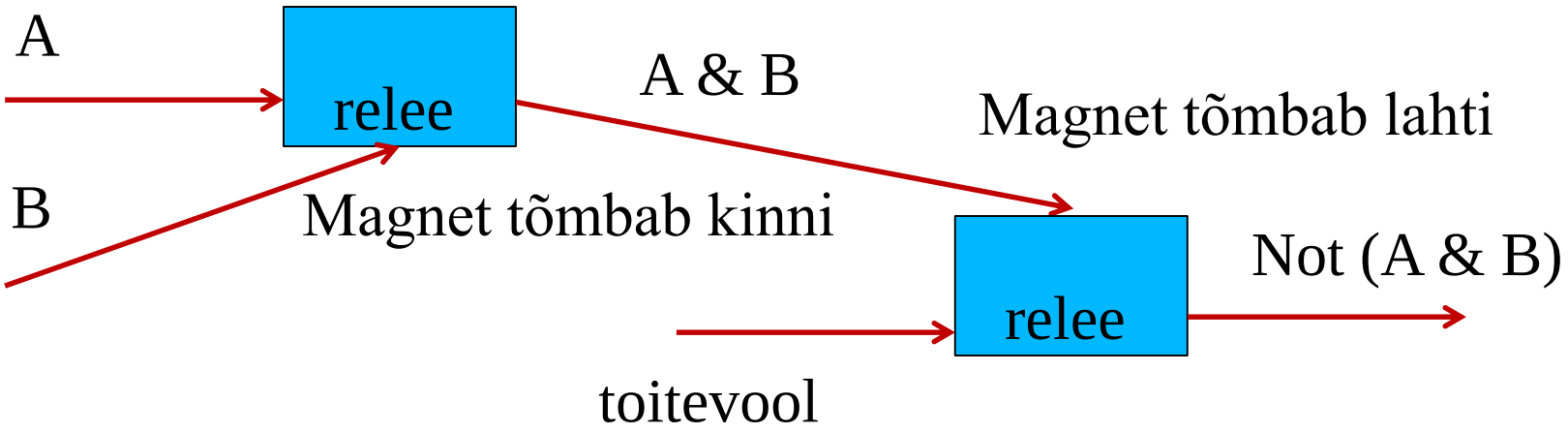


Vool läheb läbi

Magnetile
antakse
vool

Miks releed? Näide.

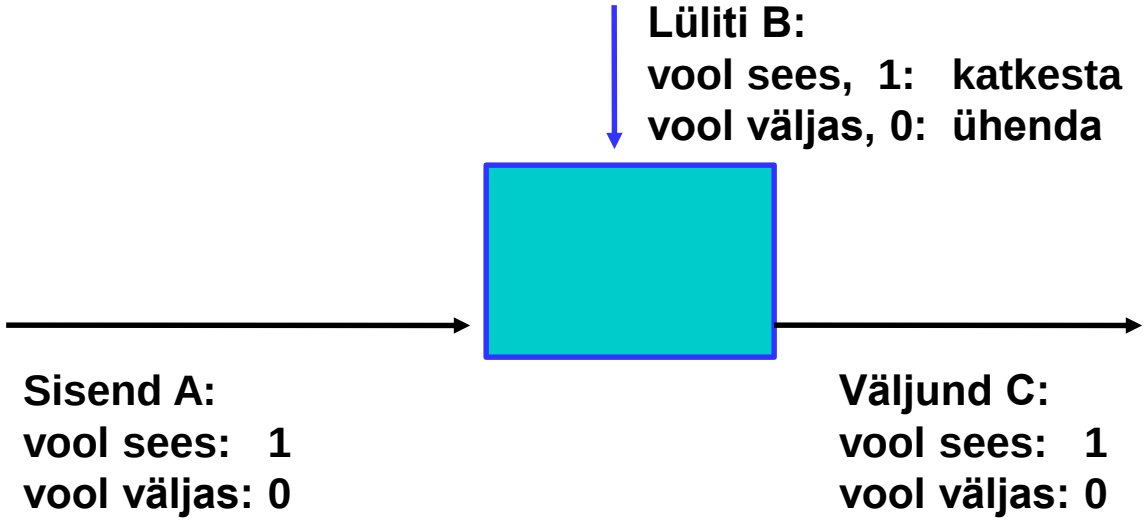
Realiseerime lausearvutuse valemi Not (A & B) releedega:



Keerukamaid asju, näiteks liitmist ja korrutamist, ehitatakse suure hulga lausearvutuse tehetena umbes samal moel.

Komponendid

- Peamine idee: **transistorid kui “katkestusmootoriga” lülitid**

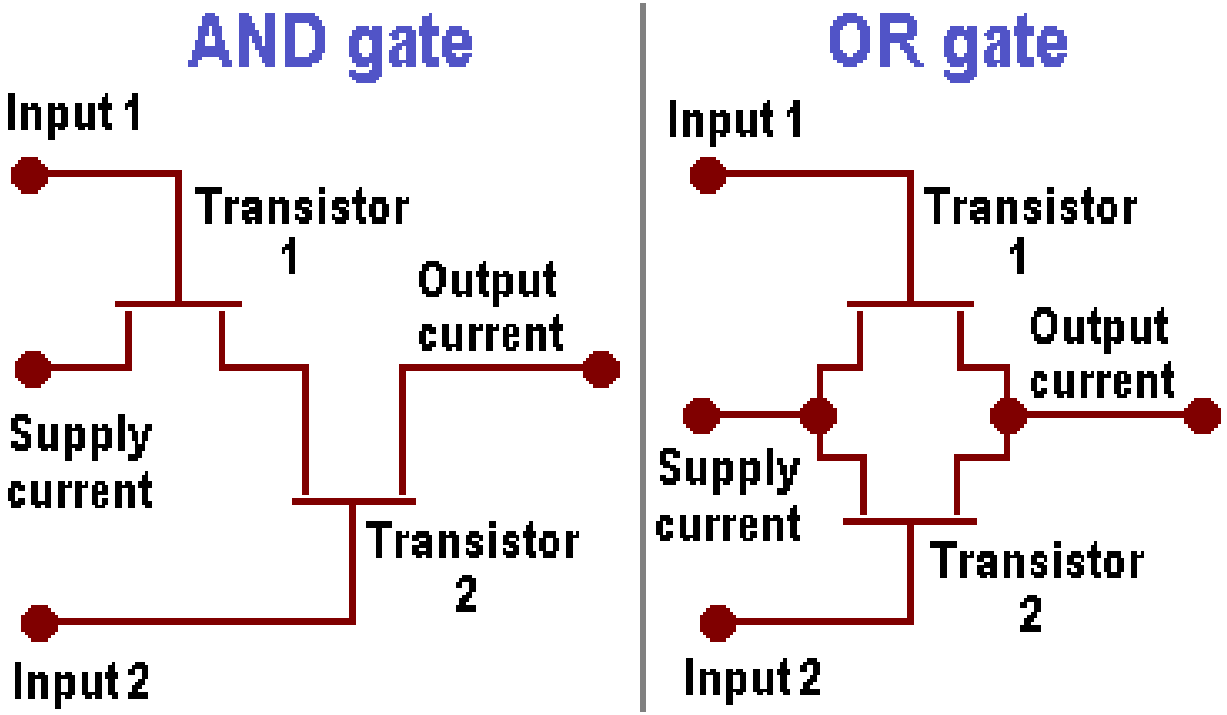


C = (A and -B)

- Väikestest komponentidest ehitatakse suuremaid, millest omakorda veel suuremaid.
- Komponendid on kui mustad kastid: teame nende väljundit vastava sisendi korral, aga enamasti mitte nende tehnilist sisu.

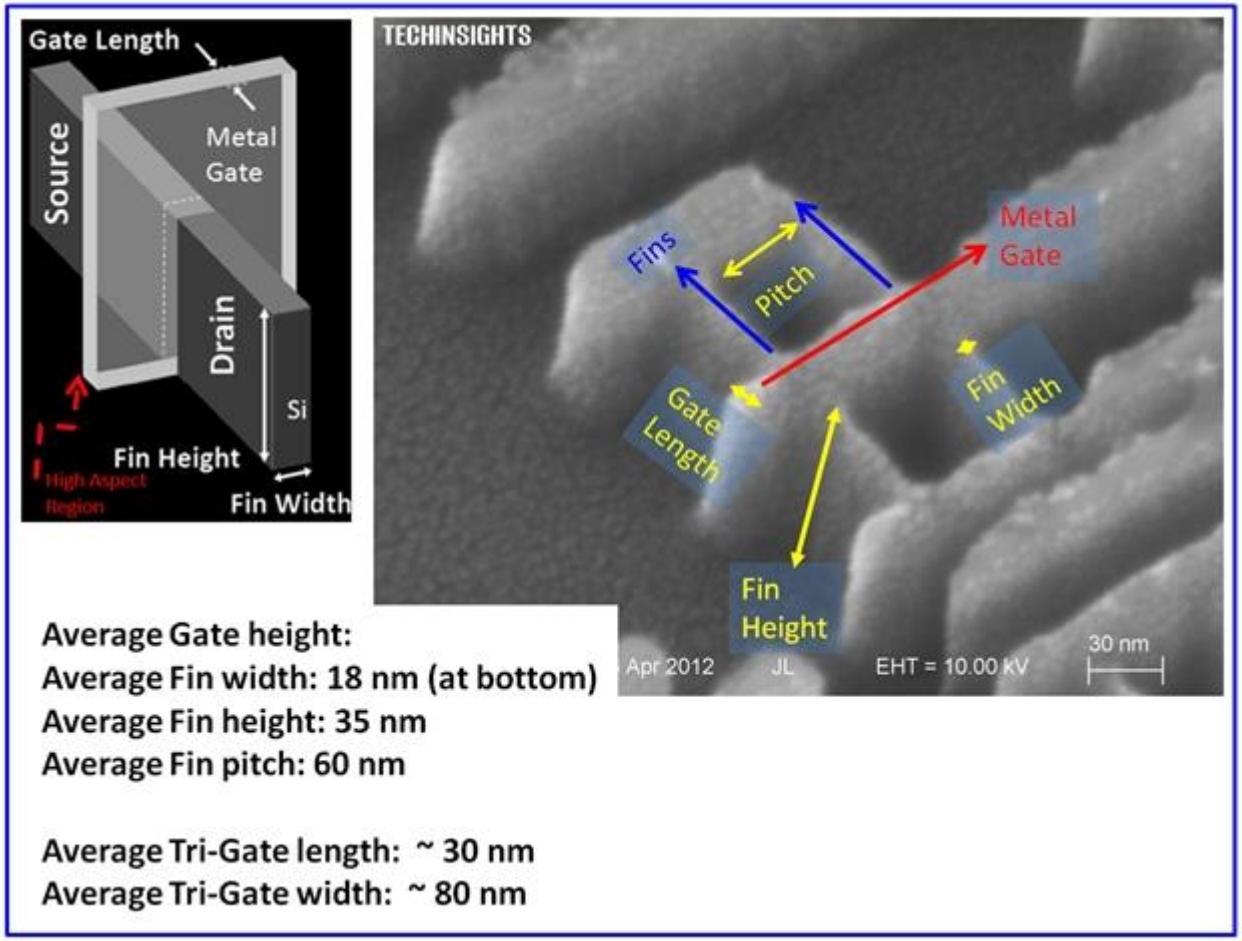
Logic gates: simple idea

- MOSFET: metal-oxide semiconductor field-effect transistor
- Input near 2V: circuit on. Input near 0V: circuit off.



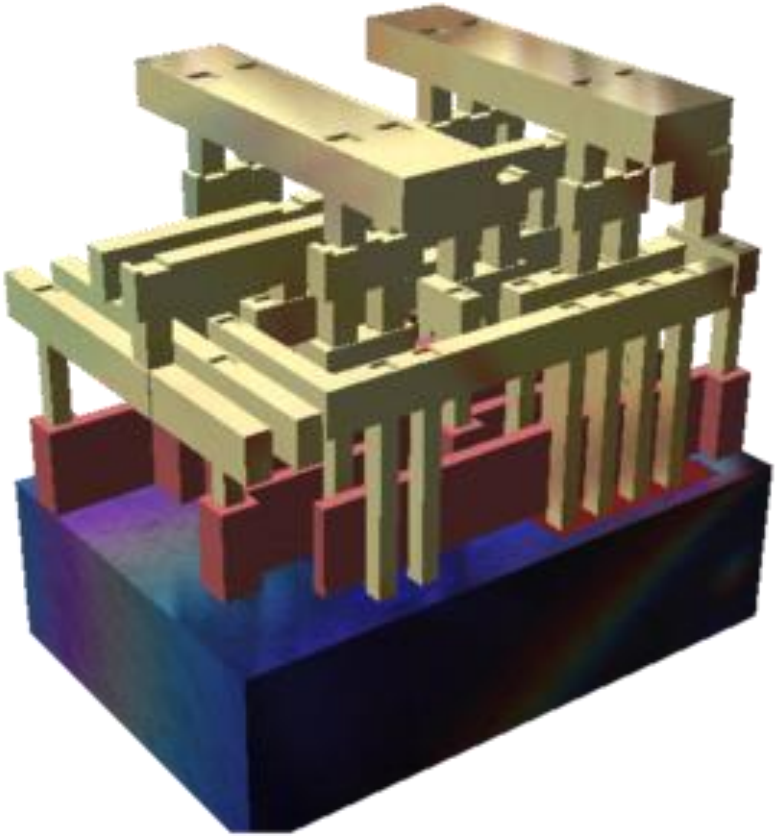
Mikroprotsessori sees

- Photo from Intel CORE i5-3550 processor: Ivy Bridge 22 nm process technology



Layers in an integrated circuit

- Blue: silicon bulk
- Red: polysilicon gates
- Yellow: metal wires

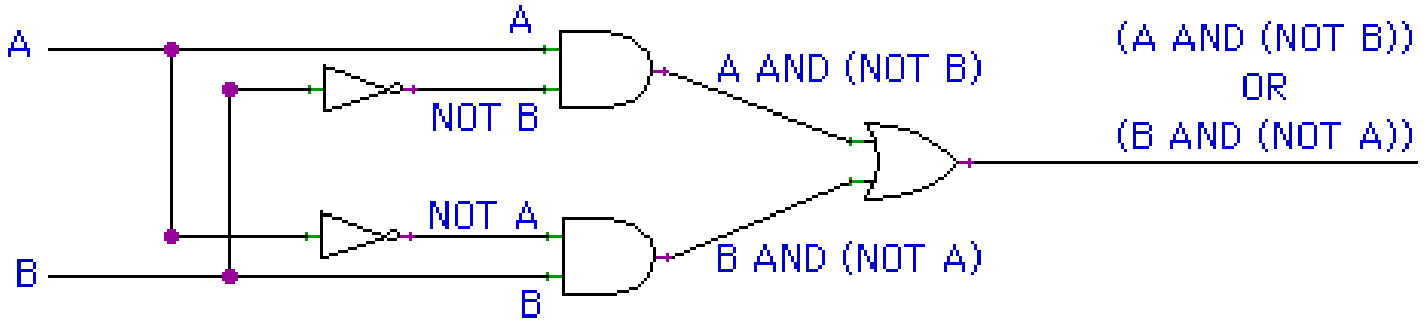


Komponendid (Eck)

NB! Loe ja harjuta nende simulaatoritega:

- http://math.hws.edu/TMCM/java/labs/xLogicCircuitsLab1.htm
- http://math.hws.edu/TMCM/java/labs/xLogicCircuitsLab2.html

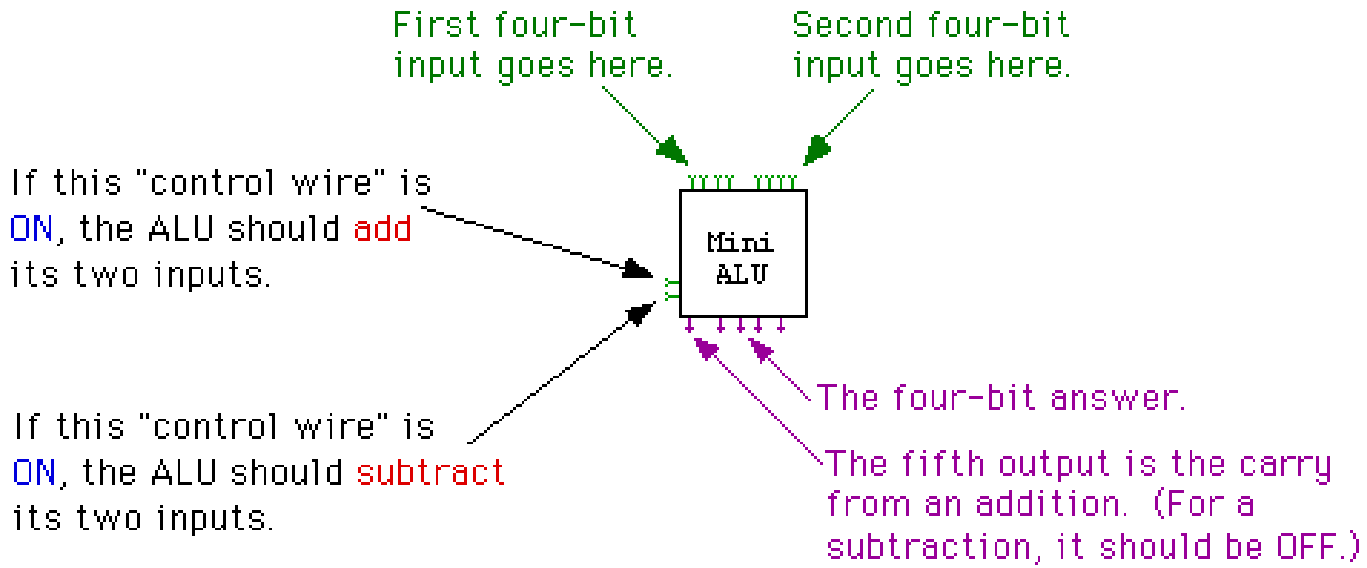
(A and (not B)) or (B and (not A))



Neljabitine liitja (four-bit adder)

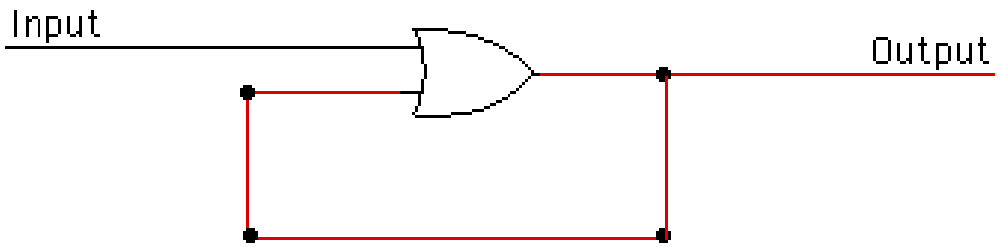
- Kaheksa pluss kaks sisendjuhet, neli pluss üks väljundjuhet

```
 1011        1111        1111        1010        0111        0001
 0110        0001        1111        0101        1010        0011
-----       -----       -----       -----       -----       -----
10001       10000       11110       01111       10001       00100
```

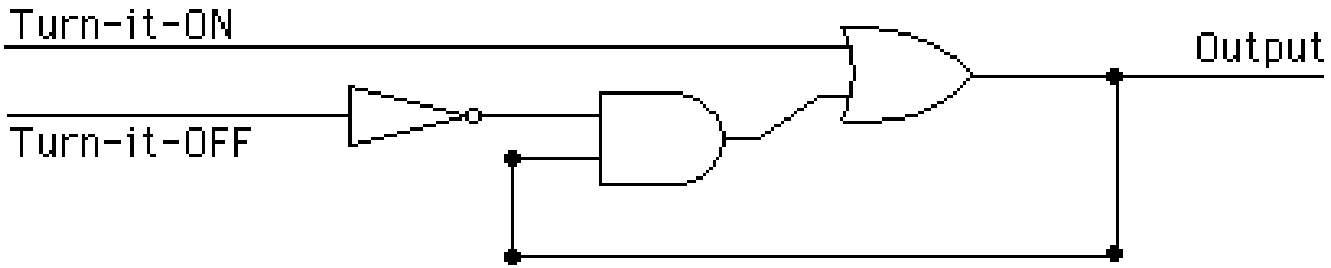


Mälu põhi-idee

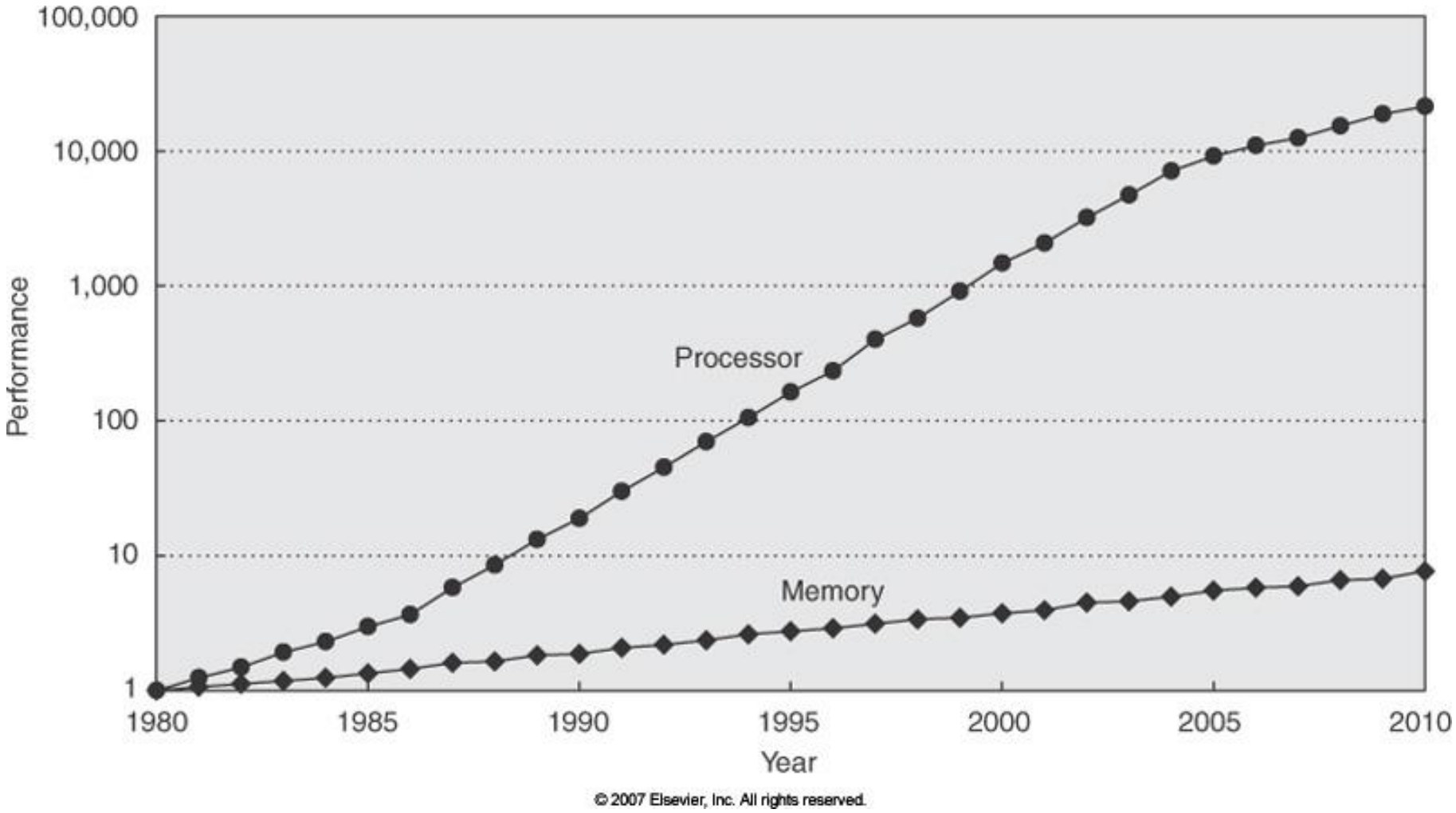
- Tagasiside



- Lülitatav tagasiside: triger



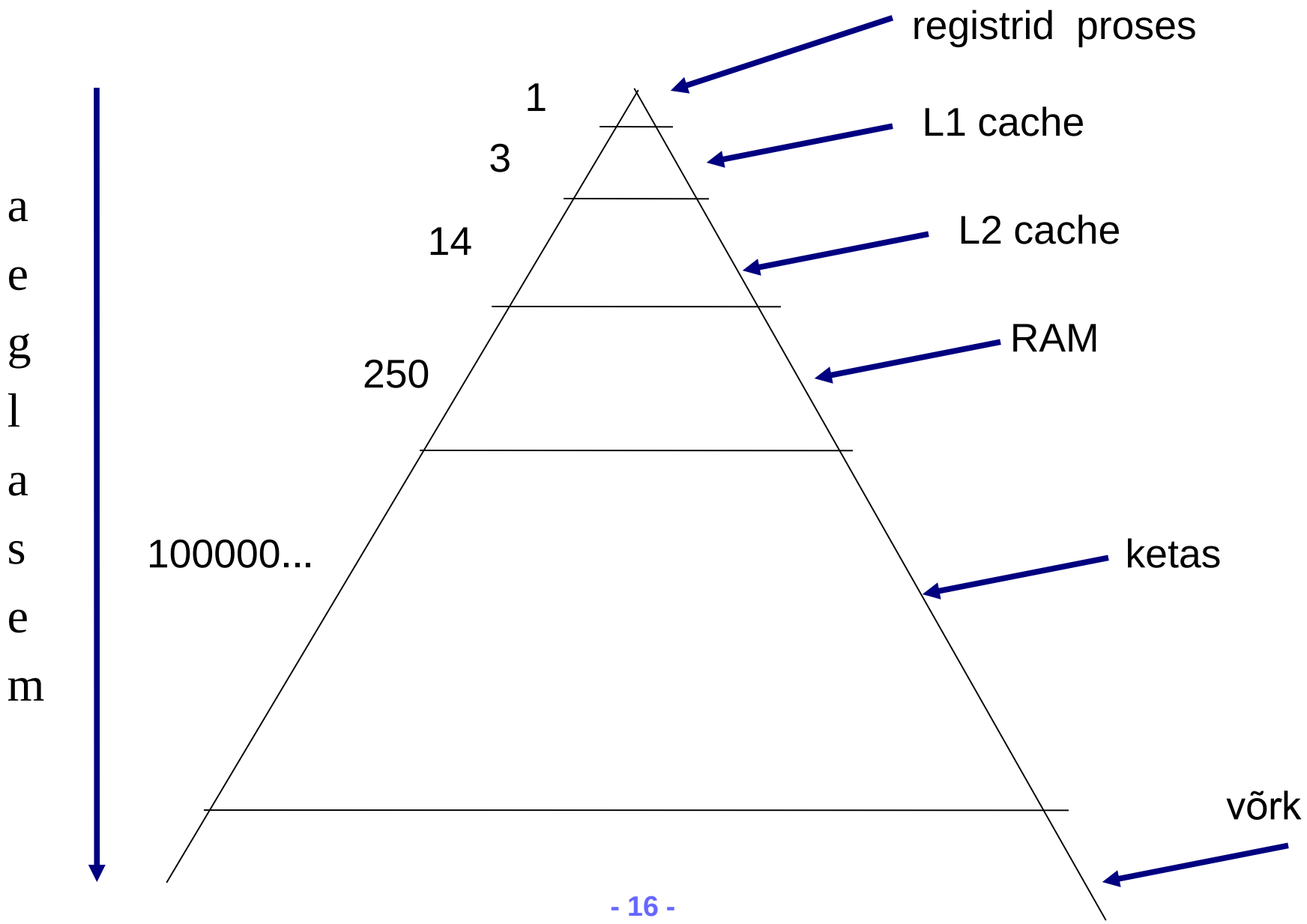
Mälu on aeglane ja ei lähe eriti ruttu kiiremaks



Eri mäluosad ehitatud eri tehnoloogiatega

- Arvuti põhimälu DRAM on ehitatud väikestest kondensaatoritest: iga kondensaator hoiab ühte bitti. Tema juhtimiseks on kasutusel veel üks transistor.
  - Kondensaatoritega mälu ei osata väga kiireks teha.
  - Kondensaatoritega mälu on odav.
- Protsessori registrid ja cache mälu on ehitatud transistoridest (SRAM): 4 või 6 transistori ühe biti kohta.
  - See on kiire tehnoloogia.
  - Samas on see kallis.
- SSD (flash mälu, mälupulgad) on samuti transistoridest, aga aeglane ja odav.

Cache ja mälu hierarhia suhteliste kiirustega



Tüüpiline lihtsat sorti protsessor

- Protsessori sees on väike hulk spetsiaal-mälupesi (registrid)
- Tehteid saab teha ainult nende registrite vahel.
- Ei ole näiteks võimalik liita otse kahte mälus olevat arvu: enne tuleb nad registritesse kopeerida, siis seal liita, siis tulemusregistrist (nn akumulaator) mällu kirjutada.
- Koha, kust mälust loetakse/kirjutatakse näitab ADDR register.
- Koha, kust lugeda järgmine käsk, näitab PC (program counter) register.

Käskude täitmine elektroonika poolt tsükliliselt

- **Lihtsal protsessoril on kaks tsüklit üksteise sees:**

- **Välimine tsükkel** suurendab igal ringil PC-d (program counterit), st igal ringil võetakse täidetav käsk järgmisest mälupesast.

  - **Sisemine tsükkel** toimub iga käsu sees. Sisemise tsükli jooksul täidetakse käsu sisemisi pisi-samme.

    Üks pisi-samm vastab mingile juhtmele voolu peale andmisele, mispeale käivitub vastav loogika-ahel protessoris ja selle tulemus salvestatakse mõnda registrisse.

- **Masina taktsagedus** on see sagedus, kui tihti pisi-samme täidetakse. Iga järgmise pisi-sammu alustamise jaoks on masinas kell, mis annab kindla sagedusega impulsse. Pisi-sammu number saadakse nende impulsside kokkulugemisega.

Keeruline protsessor on igatpidi paralleelne

Keerulisel (st kaasaegsel) protsessoril on mitu kihti parallelismi:

- Protsessoril on mitu tuuma: igaüks on sõltumatu omaette töötav protsessor.
- Üks tuum püüab (ja tihti suudab) täita hulka käske paralleelselt.
- If-lausete puhul püüab protsessor paralleelselt ette täita ühte haru, mida ta peab tõenäolisemaks.
- Cache salvestab erinevaid hiljuti kasutatud põhimälu juppe kiires SRAM mälus.
- Põhimure on minimeerida põhimälu poole pöördumisi.

Uurime reaalset protsessorit: 6502

- MOS 6502 oli kasutusel Apple II ja enamuse varaste koduarvutite protsessorina. Tal on üpris lihtsad käsud.
- Hobikasutajate ja eri-huvi-vajaduste jaoks toodetakse seda siiamaani, küll mitte enam sama firma poolt.
- Kaasaegsed protsessorid on palju keerukama käsusüsteemiga, kuid põhi-ideed on analoogilised.
- Tutvume käskude ja assembleriga pikemalt saidil

http://skilldrick.github.io/easy6502/

Meie näiteprogramm: summeerime arve 1 .... N

```
int main(int argc, char **argv) {

  int n=10;
  int i,sum=0;

  printf("Tere!\n");

  for(i=0; i<=n; i=i+1)
    sum = sum + i;

  printf("summa on %d\n",sum);
}
```

Sumto MIPS-I (SGI spinoff) assembleris

- Argumendid registritesse $4 ja $8
- Resultaat registrisse $2

```
sumto:                      ; Register $4 on n
        li    $3, 0         ; Register $3 on summa
        li    $2, 0         ; Register $2 on i
        blt   $4, $0, L3    ; Kui n<0 mine L3
L5:     addu  $3, $3, $2    ; sum = sum + i
        addu  $2, $2, 1     ; i = i + 1
        ble   $2, $4, L5    ; Kui i<=n mine L5
L3:     move  $2, $3        ; Sum sisaldab resultaati.
        Jr    $31           ; Mine aadressile registris $31
```

Sumto ja Sun Sparc'i assembler

- Sparc saadab argumendid registrites %o0 kuni %o7 ja resultaadi %o0
- Instruktsioon peale hüpet tehakse alati

```
_sumto:                     ; Register %o0 on n.
        mov %o0,%g3         ; Salvesta n registrisse %g3.
        mov 0,  %o0         ; Register %o0 on nyyd sum.
        cmp %o0,%g3         ; Kui 0>n ...
        bg  L3              ; ... mine L3
        mov 0,  %g2         ; ,aga enne i=0.
        add %o0,%g2,%o0     ; sum = sum + i.
L5:     add %g2,1  ,%g2     ; i = i + 1.
        cmp %g2,%g3         ; Kui i<=n ...
        ble,a L5            ; ... mine L5
        add %o0,%g2,%o0     ; ,aga enne sum = sum + i.
L3:     retl                ; Valmis...
nop                         ; ,aga enne ära tee midagi!
```

Sumto ja Intel 386, 486, Pentium, ...

- 386 on vähe registreid, argument saadetakse hariliku mälu kaudu. Resultaat saadetakse registris %edx.

```
_sumto:
        pushl  %ebp             ; Loome ''framepointer''-i
        movl   %esp,%ebp        ;
        movl   8(%ebp),%ecx     ; Võta n.
        xorl   %eax,%eax        ; sum = 0
        xorl   %edx,%edx        ; i = 0
        cmpl   %ecx,%eax        ; Kui i>n ...
        jg L3                   ; ... mine L3
        .align 2
L5:     addl   %edx,%eax        ; sum = sum + i
        incl   %edx             ; i = i+1
        cmpl   %ecx,%edx        ; Kui i<=n ...
        jle L5                  ; ... mine L5
L3:     leave                   ; Taasta ebp.
        ret                     ; Valmis!
```

Sumto ja ARM

- ARMil on 16 registrit r0, ..., r15, kuid osa neist on eritähendusega (näiteks, r15 on program counter)

```
        entry
sumto:
        mov r1, #0              ; sum = 0
        mov r2, #0              ; i = 0
        ldr r3, N               ; r3=N
loop:
        cmp r3, r2              ; eeldame r3=N
        bgt finish              ; kui n>i, mine finish
        add r2, r2, #1          ; i = i + 1
        add r1, r1, r2          ; sum = sum + i
        B loop                  ; uus tsükkel
 finish:
        end
N       DCD  #1000              ; mälupesa N algväärtusega
```

Uurime C kompilaatorit

- Võtame eelmise 0 ... N summa näite C keeles ja vaatame, mida kompilaator teeb.

Kompileeritud programmi valmimine ja käivitamine

Olgu meil (näiteks C keeles) failid main.c ja swap.c

Teeme gcc main.c swap.c -o minuprogramm

Kompilaator (näiteks gcc) teeb järjest mitut eri asja:

- **Kompileerimine**
  - Kompilaator teeb neist assemblerikeelsed ajutised failid
  - Kompilaator teeb assemblerfailidest masinkood+sümbolinfo failid
- **Linkimine**
  - Linkur otsib kokku vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega

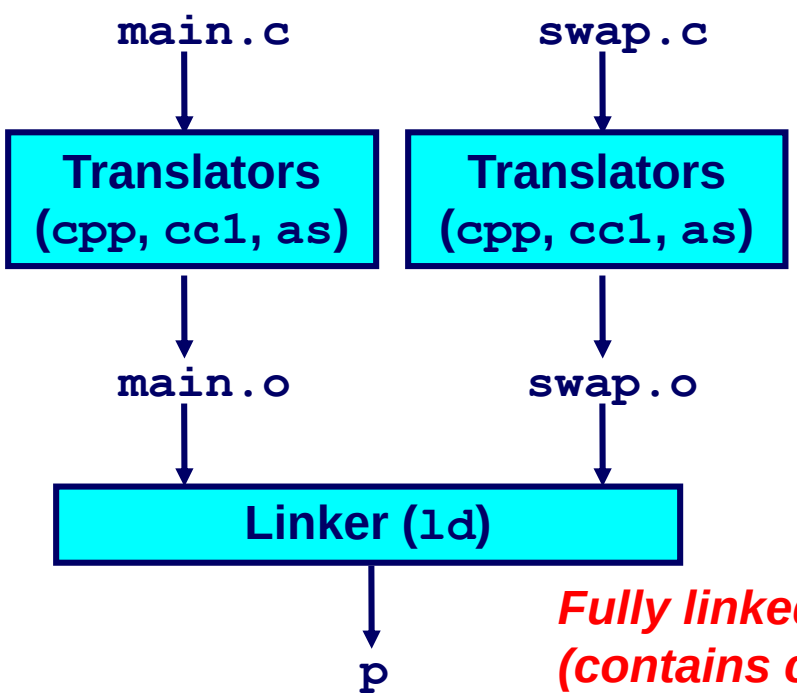
Käivitame saadud programmifaili minuprogramm:

- Opsüsteemi **loader** otsib lisaks vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega
- Saadud kogum paigutatakse mällu, tehakse opsüsteemi infoblokk tema jaoks (protsess) ja kogum käivitatakse

(CMU Bryant & Hallaron course) Static Linking

Programs are translated and linked using a *compiler driver*:

- `unix> gcc -O2 -g -o p main.c swap.c`
- `unix> ./p`

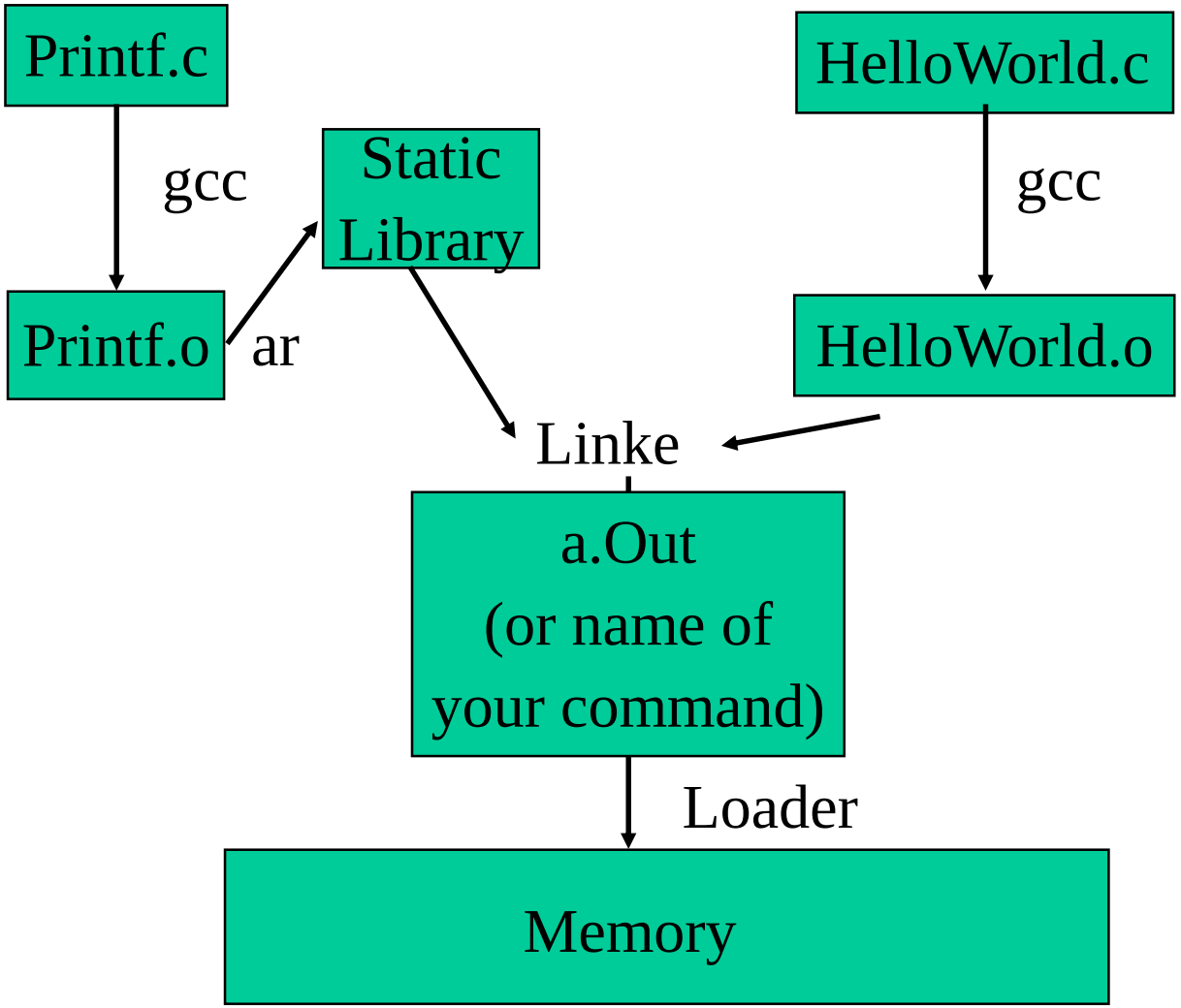


Source files

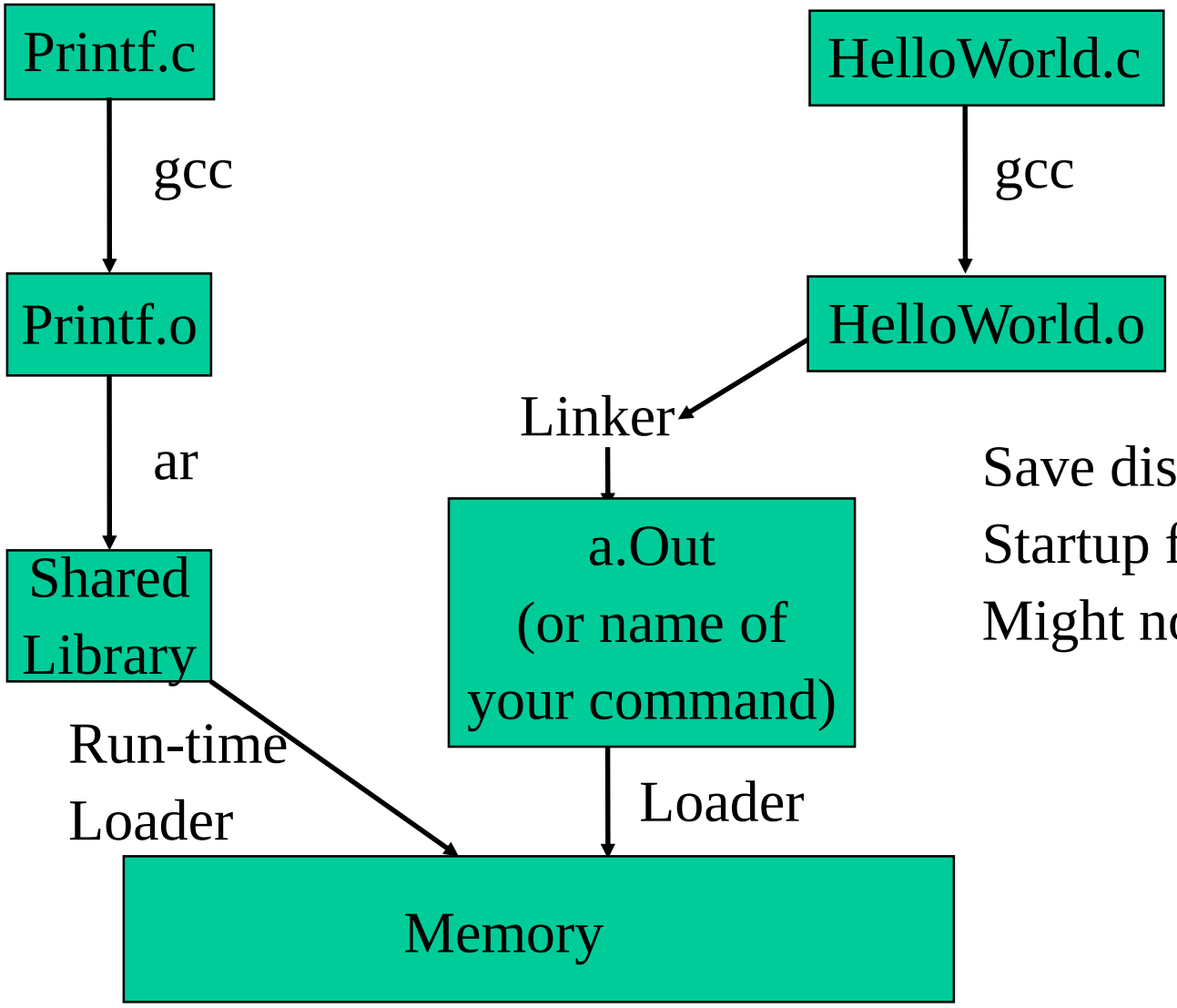
Separately compiled relocatable object files

Fully linked executable object file (contains code and data for all functions defined in main.c and swap.c

(CMU B&H ..) Static Linking and Loading



(CMU B&H ..) Run-time Linking/Loading



Save disk space.

Startup faster.

Might not need all.